# Tutorial 6 - Feature point descriptors RANSAC

Gidon Rosalki

2025-11-26

## 1 Feature descriptors

We have discussed how to detect feature points from both images, and how to use these pairs to align images, and we are now going to discuss how to build a descriptor from each point. These descriptors will help us match between the points that we have detected, and are realistically a description of the area *around* said point. This point descriptor should be both *invariant* and *distinctive*.

The simplest descriptor (which we will not use) would be a vector of image pixels, where we can find similarity between them through Euclidean distance, or cross correlation. This is not invariant to rotation, or any other modification other than translation, which is eminently problematic. We could also measure through local histograms, gradients, and so on, but realistically these descriptors are too imprecise, due to not being invariant to sufficient properties.

### 1.1 MOPS - Multi Scale Oriented Patches

MOPS makes use of multi scale Harris corners, and uses regions taken from a blurred image. It is also (usefully) geometrically invariant to rotation and scaling. The descriptor vector uses normalised sampling of a local patch (size 8x8), and is photometrically invariant to changes in intensity.

The MOPS description vector is attained by taking the orientation of the points found by Harris. The points are inside square of 40x40. We then acquire the orientation by taking the gradient of the (blurred) image at that point. We then take a patch by going up 2 levels in the pyramid (meaning that we are taking every 5th pixel), and we create a descriptor with $x, y, s$ ($s$ is scale) from Harris, and the orientation ($\theta$) from this operation.
The 8x8 oriented patch, is sampled at a lower resolution, and normalised with

$$I' = \frac{I - \mu}{\sigma}$$

We use the Euclidean distance as our distance function.

We then measure similarity by the ratio between the first and second Nearest Neighbour. If there is an incorrect matching between our point, and the first NN, then there will be a similar value as a result of the distance to the second NN, resulting in a fairly high ratio, telling us to not match these points. However, when this value increases, it means that there is (probably) a correct matching between this point and the nearest neighbour.

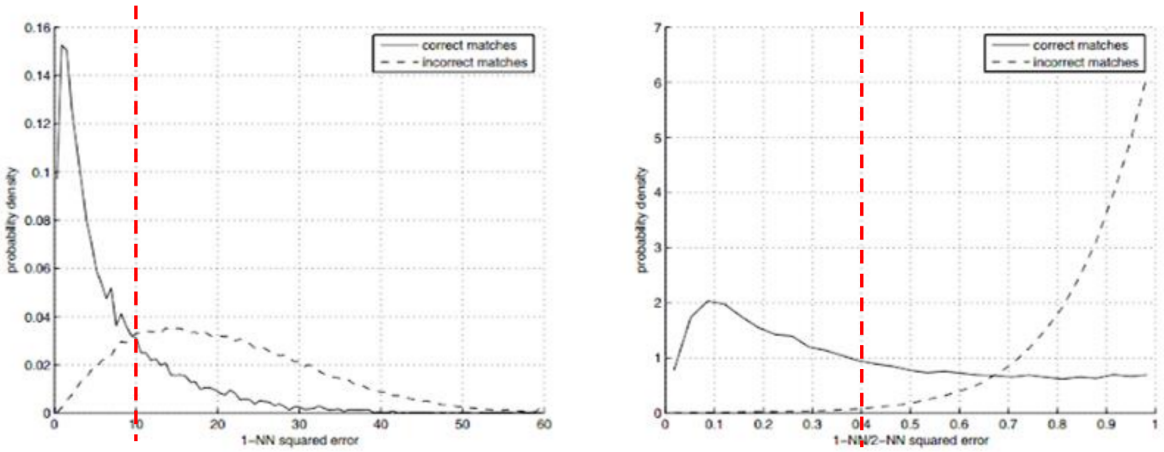


Figure 1: Nearest Neighbours ratio

### 1.2 SIFT - Scale Invariant Feature Transform

This makes use of difference of Gaussians local extrema, and the orientation of the histogram of gradient directions. The descriptor vector is built from a histogram of local gradient magnitudes, of size $4 \times 4 \times 8 = 128$.

Image content is transformed into local feature coordinates that are invariant to translation, rotation, scale, and other such parameters. To do this, it does not work through Harris point finding, but rather through computing the

Difference of Gaussian (DOG) pyramid (Burt & Adelson, 1983). This measures the differences between different scales, one octave at a time. Next, it finds matching areas between adjacent scales (up to the distance of 8 pixels), and finds matching points within a certain threshold since noise can also produce matches. Yes, this is a messy description, we are not expected to understand all its details.

Once we have the features from SIFT, we can create a histogram of local gradient directions, quantised into 36 bins (each bin is 10 degrees). We weight each point with the Gaussian window (as in, closer to the middle of the Gaussian increases the weight, and closer to the edge reduces it), and the magnitude, and assign a canonical orientation at a peak of the smoothed (so each point spreads a little to the nearest bins as well) histogram. Each key specifies stable 2D coordinates ($x$, $y$, scale, and orientation).
The highest bar in the histogram will then be the direction of our patch. This has been essentially a long and complicated method of finding the direction of a point.

To now build the actual SIFT descriptor, we take the gradient magnitude, and orientation, computed over the 16 by 16 array of locations in scale space around the key point. We rotate these to align with our point, and re-split it into an array of 4 by 4 patches (each containing 16 pixels). We take histograms of the directions of each of these 4x4 pixel windows, but this time split into 8 bins rather than 36. Additionally, the pixels on the edge of each cell will also slightly affect the directions of the neighbouring cell, and we will also smooth slightly between the bins of the histogram (as we did previously). So we have 8 bins in each histogram, and 16 histograms, and from this we will connect them into a large vector, of $8 \cdot 16 = 128$ dimensions. This vector will then be normalised (its magnitude is now 1), and if any of the resulting values are greater than 0.2, we will reduce them, and renormalise the vector.

There are some advantages of invariant local features:

- Locality: The features are local, and so are robust to occlusion and clutter
- Distinctiveness: Individual features can be matched into a large database of objects
- Efficiency: There is close to real time performance
- Extensibility: It can be easily extended into a wide range of differing feature types, each adding robustness.

The locality point builds into 3D object recognition, since we can recognise objects even when occluded.

### 1.3 Differences

| | MOPS | SIFT |
|---|---|---|
| Feature Point Detection | Scale invariant Harris $(x, y, s)$ | Difference of Gaussians local extrema $(x, y, s)$ |
| Orientation | Blurred gradient direction $(x, y, s, \theta)$ | Histogram of weighted gradient directions $(x, y, s, \theta)$ |
| Patch + scale invariance | $8 \times 8$ pixels from levels $s + 2$ in the pyramid ($\approx 40 \times 40$ on level $s$) | $16 \times 16$ gradient from scale $s$, separated to $4 \times 4$ histograms of 8 orientations |
| Rotation invariance | Path taken at orientation $\theta$ | Orientation $\theta$ subtracted from gradients direction |
| Intensity invariance | patch $= \frac{\text{patch}-\text{mean}}{\text{STD}}$ | Normalise to unit vector and clip at 0.2 |
| Distance metric | $\frac{1-NN}{2-NN}$ (Euclidean) | $\frac{1-NN}{2-NN}$ (Euclidean) |

Table 1: Comparison table

# 2 RANSAC

Out of the necessary steps:

1. Detect feature points in both images
2. Build a descriptor from each point
3. Find corresponding pairs
4. Use these pairs to align images

We have achieved 1, 2, and 4. Let us now discuss 3, finding the corresponding pairs. We will assume that given two images, we have computed feature points, and their descriptors in both images. For each feature point in each image we computed the $k$ good matches in the other image $k = 1, 2, \ldots$ (e.g. exhaustive search). We selected candidate matching pairs. E.g. two points are a matched pair only if each is in the top $k$ matches of the other point.

We need to figure out how to overcome the wrong matches. To do this, we need robust methods.

## 2.1 Robust methods

The goal of many algorithms is parametric model fitting. Data measured in real images is *inaccurate*, thanks to noise, occlusions, ambiguity, or perhaps wrong feature extraction. The goal of **robust methods** is to tolerate outliers.

A common concept is to use the Least Squared error minimisation. However, despite this working very well for a set of clean points, this can be very severely impacted by outliers. Therefore, LSE is **not** robust. It is good since it is a very clear objective function, and easy to optimise, but it is sensitive to outliers, and cannot find multiple matches.

## 2.2 RANdom SAmple Consensus

In this algorithm, we estimate parameters of a model by random sampling of observed data. The algorithm is as follows:

1. Sample (randomly) the number of points required to fit the model
2. Solve for model parameters using samples
3. Score the fraction of inliers within a preset threshold of the model
4. Repeat 1 - 3 until the best model is found with high confidence

So, how many iterations are needed? We want to ensure that if we run for inliers iterations, in probability $p$, we choose $s$ inliers in some iterations (for example, $p = 99\%$). We denote inliers by probability $\omega$. Thus, to find $n$

$$(1-\omega^s)^n \leq 1-p \implies n \geq \frac{\log(1-p)}{\log(1-\omega^s)}$$

### 2.2.1 Parameter selection

- $s$: Number of points needed to fit the model
- $\omega$: The probability of finding an inlier
- $\omega^s$: Probability that all $s$ points are inliers
- $1-\omega^2$: Probability that at least one point is an inlier
- $p$: Probability of choosing $s$ inliers in some iterations
- $1-p = \left(1-\omega^2\right)^n$: Probability to never select $s$ inliers
- $n$: Number of RANSAC iterations

So:

$$n = \frac{\log(1-p)}{\log(1-\omega^s)}$$

### 2.2.2 No assumption of outlier proportion

We may also alter the algorithm, since $\omega$ is often unknown a priori, so we can pick the worst case (e.g. 50%), and adapt if more inliers are found:

**1**

$N = \infty$
sample_count $= 0$
**while** n $>$ sample_count **do**
    Choose a sample
    $\omega 1 = \frac{\#\text{inliers}}{\#\text{points}}$
    **if** $\omega 1 > \omega$ **then**
        Re-compute $n$ from $\omega$
    **end if**
    Increment the sample_count by 1
**end while**
Terminate

### 2.2.3 Conclusions

RANSAC is good due to its robustness to outliers, and is suitable for many model fittin gcases, while being easy to implement. However, it is not so good since the computational time grows quickly with the fraction of outliers, and the number of parameters.

## 2.3 Back to Homography Estimation

The RANSAC loop is as follows:

1. Randomly select 4 feature pairs
2. Compute homography $H$ exactly
3. Compute the inliers where $D\left(p_i', Hp_i\right) < \varepsilon$
4. Keep the largest set of inliers
5. Re-compute $H$ using least squares on all inliers in the largest set (can use LSE since we have removed outliers)